LR Parsing

Mattox Beckman

beckman@iit.edu

Illinois Institute of Technology

LL parsing is easy to use, but unless the grammar is *just right*, you may need to perform all kinds of tedious[a] transformations. For this reason, it is much more common to use LR parsing.

- Be able to explain how LR parsing is different than LL parsing.
- Know how to generate the Characteristic Finite State Automata for a grammar.
- Know what the item sets mean.
- Know how to create the LR Action and Goto tables
- Know what a Shift-reduce conflict is, and how to fix it
- Know what a reduce-reduce conflict is, and how to fix it
- Know how to use the parse tables to generate an “execution trace” of a parse.

An LR(1) Parser

- scans the text from **L**eft to right,
- produces a **R**ightmost derivation,
- using **1** token of lookahead.

Contrast to an LL(1) Parser

- scans the text from **L**eft to right,
- produces a **L**eftmost derivation,
- using **1** token of lookahead.

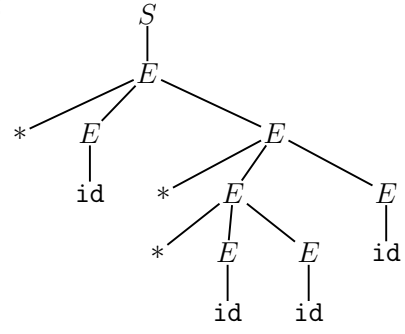
Consider the grammar $S \to E$

$E \to id$

$E \to * \; E \; E$

and the input `* id * * id id id`.

Resulting parse tree:



$S \rightarrow E$
$E \rightarrow id$
$E \rightarrow * \; E \; E$

Input: `* id * * id id id`

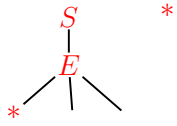
LL Parse **LR Parse**

LR Parsing
Dueling Parsers
February 14, 2007
§1 **LR Grammars**

$S \rightarrow E$
$E \rightarrow id$
$E \rightarrow * \; E \; E$

Input: `* id * * id id id`

LL Parse **LR Parse**



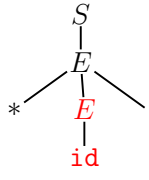
Illinois Institute of Technology Mattox Beckman
LR Parsing – p. 5

LR Parsing
Dueling Parsers
February 14, 2007
§1 **LR Grammars**

$S \rightarrow E$
$E \rightarrow id$
$E \rightarrow * \; E \; E$

Input: `* id * * id id id`

LL Parse **LR Parse**



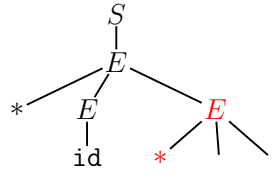
Illinois Institute of Technology Mattox Beckman
LR Parsing – p. 5

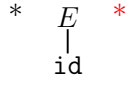
LR Parsing
Dueling Parsers
February 14, 2007
§1 **LR Grammars**

$S \rightarrow E$
$E \rightarrow id$
$E \rightarrow * \; E \; E$

Input: `* id * * id id id`

LL Parse **LR Parse**



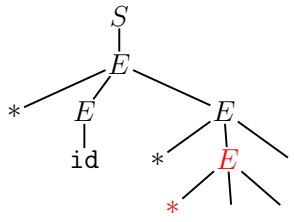


Illinois Institute of Technology Mattox Beckman
LR Parsing – p. 5

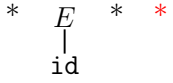
$S \rightarrow E$
$E \rightarrow id$
$E \rightarrow * \; E \; E$

Input: `* id * * id id id`

LL Parse



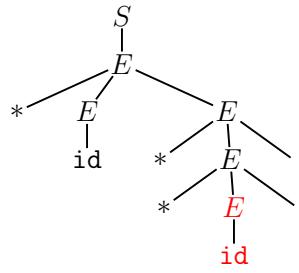
LR Parse



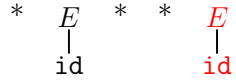
$S \rightarrow E$
$E \rightarrow id$
$E \rightarrow * \; E \; E$

Input: `* id * * id id id`

LL Parse



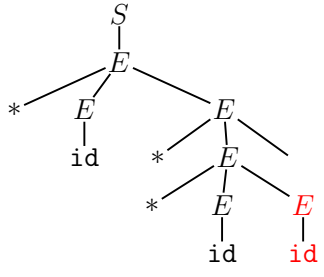
LR Parse



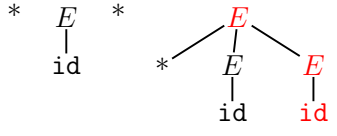
$S \rightarrow E$
$E \rightarrow id$
$E \rightarrow * \; E \; E$

Input: `* id * * id id id`

LL Parse



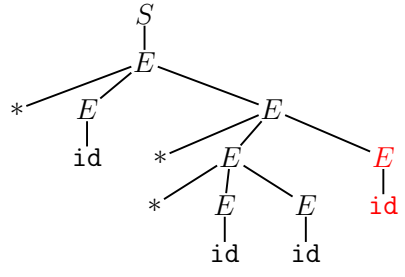
LR Parse



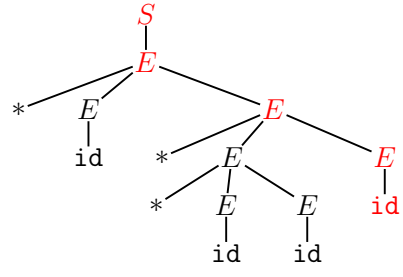
$S \rightarrow E$
$E \rightarrow id$
$E \rightarrow * \; E \; E$

Input: `* id * * id id id`

LL Parse



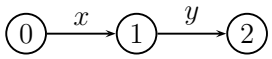
LR Parse



The Goal

- We want to use a *state machine* to handle the LR parse for us.

Consider the simple grammar $A \rightarrow x\ y$. The resulting state machine is:



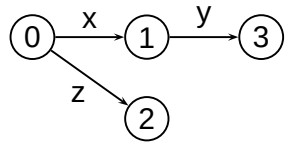
These states correspond to different stages of the production.

0. $A \rightarrow \bullet\ x\ y$
1. $A \rightarrow x\ \bullet\ y$
2. $A \rightarrow x\ y\ \bullet$

The "•" is called "dot" or "the cursor".

Multiple Productions

Consider the grammar $A \rightarrow x\ y$, $A \rightarrow z$

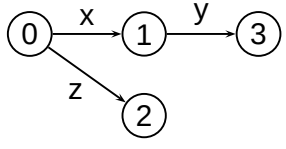


0. $A \rightarrow \bullet\ x\ y$
   $A \rightarrow \bullet\ z$

To start, place the cursor at the beginning of the A productions. This represents the beginning when we've received no input. You need to include both productions here, since we don't know which of the two productions we will use.

Multiple Productions

Consider the grammar $A \rightarrow x\ y$, $A \rightarrow z$

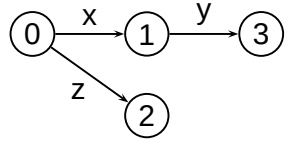


0. $A \rightarrow \bullet\ x\ y$ ⇐
   $A \rightarrow \bullet\ z$
1. $A \rightarrow x\ \bullet\ y$

If you are in state 0 and input an x, you will advance the cursor past the x to get state 1.

Multiple Productions

Consider the grammar $A \rightarrow x\ y$, $A \rightarrow z$

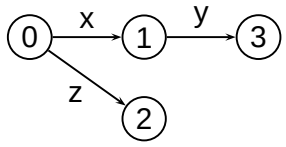


0. $A \rightarrow \bullet\ x\ y$
   $A \rightarrow \bullet\ z$ ⇐
1. $A \rightarrow x\ \bullet\ y$
2. $A \rightarrow z\ \bullet$

If you are in state 0 and input a z, you will advance the cursor past the z to get state 2.

Consider the grammar

$A \rightarrow x\ y$
$A \rightarrow z$



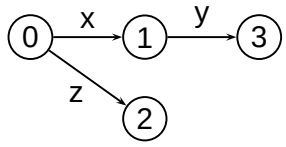
0. $A \rightarrow \bullet\ x\ y$
   $A \rightarrow \bullet\ z$
1. $A \rightarrow x\ \bullet\ y \Leftarrow$
2. $A \rightarrow z\ \bullet$
3. $A \rightarrow x\ y\ \bullet$

If you are in state 1 and input a y, you will advance the cursor past the y to get state 3.

LR Parsing
Multiple Productions
February 14, 2007
§2 **Characteristic Finite State Automaton**

Consider the grammar

$A \rightarrow x\ y$
$A \rightarrow z$

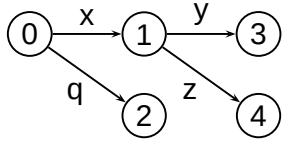


0. $A \rightarrow \bullet\ x\ y$
   $A \rightarrow \bullet\ z$
1. $A \rightarrow x\ \bullet\ y$
2. $A \rightarrow z\ \bullet \Leftarrow$
3. $A \rightarrow x\ y\ \bullet \Leftarrow$

These last two states are already complete, so no new states are formed.

Consider the grammar

$A \rightarrow x\ y$
$A \rightarrow q$
$A \rightarrow x\ z$



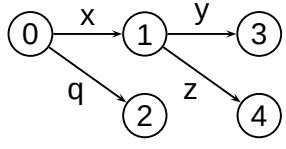
0. $A \rightarrow \bullet\ x\ y$
   $A \rightarrow \bullet\ q$
   $A \rightarrow \bullet\ x\ z$

To start, copy all the A productions and place the cursor in front.

LR Parsing
Simultaneous Movement
February 14, 2007
§2 **Characteristic Finite State Automaton**

Consider the grammar

$A \rightarrow x\ y$
$A \rightarrow q$
$A \rightarrow x\ z$

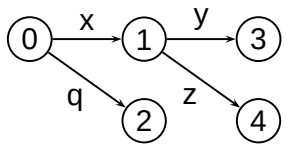


0. $A \rightarrow \bullet\ x\ y \Leftarrow$
   $A \rightarrow \bullet\ q$
   $A \rightarrow \bullet\ x\ z \Leftarrow$
1. $A \rightarrow x\ \bullet\ y$
   $A \rightarrow x\ \bullet\ z$

The transition from state 0 to 1 causes two productions to move: when we read x we could be parsing either xy or xz.

Consider the grammar

$A \rightarrow x\ y$
$A \rightarrow q$
$A \rightarrow x\ z$



0. $A \rightarrow \bullet\ x\ y$
   $A \rightarrow \bullet\ q$⇐
   $A \rightarrow \bullet\ x\ z$

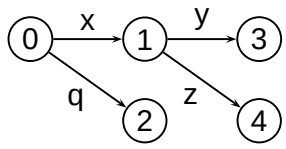
1. $A \rightarrow x\ \bullet\ y$
   $A \rightarrow x\ \bullet\ z$
2. $A \rightarrow q\ \bullet$

If we are in state 0, reading a q brings us to state 2.

LR Parsing February 14, 2007
Simultaneous Movement §2 **Characteristic Finite State Automaton**

Consider the grammar

$A \rightarrow x\ y$
$A \rightarrow q$
$A \rightarrow x\ z$



0. $A \rightarrow \bullet\ x\ y$
   $A \rightarrow \bullet\ q$
   $A \rightarrow \bullet\ x\ z$

1. $A \rightarrow x\ \bullet\ y$⇐
   $A \rightarrow x\ \bullet\ z$
2. $A \rightarrow q\ \bullet$
3. $A \rightarrow x\ y\ \bullet$

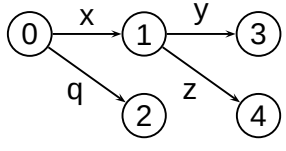
If we are in state 1, reading a y brings us to state 3.

Illinois Institute of Technology Mattox Beckman LR Parsing – p. 8

LR Parsing February 14, 2007
Simultaneous Movement §2 **Characteristic Finite State Automaton**

Consider the grammar

$A \rightarrow x\ y$
$A \rightarrow q$
$A \rightarrow x\ z$



0. $A \rightarrow \bullet\ x\ y$
   $A \rightarrow \bullet\ q$
   $A \rightarrow \bullet\ x\ z$

1. $A \rightarrow x\ \bullet\ y$
   $A \rightarrow x\ \bullet\ z$⇐

2. $A \rightarrow q\ \bullet$
3. $A \rightarrow x\ y\ \bullet$
4. $A \rightarrow x\ z\ \bullet$

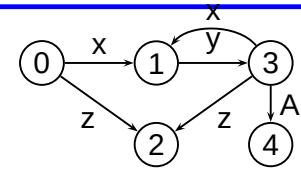
If we are in state 1, reading a z brings us to state 4. None of the remaining states are expecting input.

Illinois Institute of Technology Mattox Beckman LR Parsing – p. 8

LR Parsing February 14, 2007
Being Several Places at Once §2 **Characteristic Finite State Automaton**

Consider the grammar

$A \rightarrow x\ y\ A$
$A \rightarrow z$

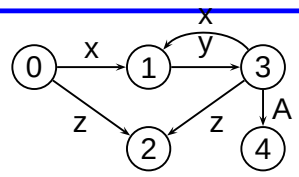


0. $A \rightarrow \bullet\ x\ y\ A$
   $A \rightarrow \bullet z$

Normal Start Sequence: Add the initial productions.

Illinois Institute of Technology Mattox Beckman

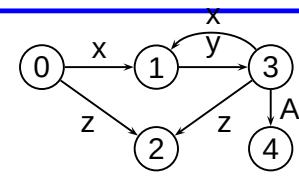
Consider the grammar $A \rightarrow x\ y\ A$
$A \rightarrow z$



0. $A \rightarrow \bullet\ x\ y\ A$ ⇐
   $A \rightarrow \bullet z$
1. $A \rightarrow x\ \bullet\ y\ A$

State 0: Shift the x to make state 1.

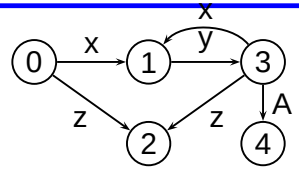
Consider the grammar $A \rightarrow x\ y\ A$
$A \rightarrow z$



0. $A \rightarrow \bullet\ x\ y\ A$
   $A \rightarrow \bullet z$ ⇐
1. $A \rightarrow x\ \bullet\ y\ A$
2. $A \rightarrow z\ \bullet$

State 0: Shift the z to make state 2.

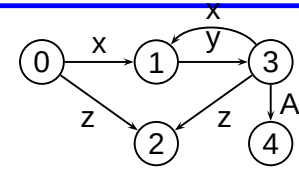
Consider the grammar $A \rightarrow x\ y\ A$
$A \rightarrow z$



0. $A \rightarrow \bullet\ x\ y\ A$
   $A \rightarrow \bullet z$
1. $A \rightarrow x\ \bullet\ y\ A$ ⇐
2. $A \rightarrow z\ \bullet$
3. $A \rightarrow x\ y\ \bullet\ A$

State 1: Shift the y to make state 3. Note: the cursor is in front of an A now.

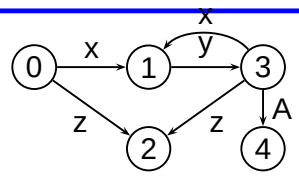
Consider the grammar $A \rightarrow x\ y\ A$
$A \rightarrow z$



0. $A \rightarrow \bullet\ x\ y\ A$
   $A \rightarrow \bullet z$
1. $A \rightarrow x\ \bullet\ y\ A$
2. $A \rightarrow z\ \bullet$
3. $A \rightarrow x\ y\ \bullet\ A$
   $A \rightarrow \bullet\ x\ y\ A$
   $A \rightarrow \bullet\ z$

Because the cursor is in front of an A in state 3, we have to add the initial items for A again. This operation is known as taking the closure of the state.

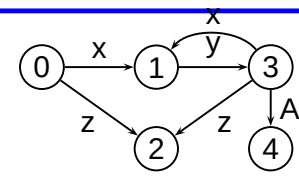
Consider the grammar $A \to x\ y\ A$, $A \to z$



0. $A \to \bullet\ x\ y\ A$
   $A \to \bullet z$
1. $A \to x\ \bullet\ y\ A$
2. $A \to z\ \bullet \Leftarrow$
3. $A \to x\ y\ \bullet\ A$
   $A \to \bullet\ x\ y\ A$
   $A \to \bullet\ z$

State 2: No input expected.

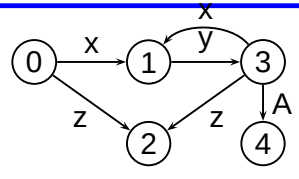
Consider the grammar $A \to x\ y\ A$, $A \to z$



0. $A \to \bullet\ x\ y\ A$
   $A \to \bullet z$
1. $A \to x\ \bullet\ y\ A$
2. $A \to z\ \bullet$
3. $A \to x\ y\ \bullet\ A \Leftarrow$
   $A \to \bullet\ x\ y\ A$
   $A \to \bullet\ z$
4. $A \to x\ y\ A\ \bullet$

State 3: Shift the A to make state 4.

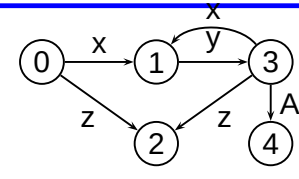
Consider the grammar $A \to x\ y\ A$, $A \to z$



0. $A \to \bullet\ x\ y\ A$
   $A \to \bullet z$
1. $A \to x\ \bullet\ y\ A$
2. $A \to z\ \bullet$
3. $A \to x\ y\ \bullet\ A$
   $A \to \bullet\ x\ y\ A \Leftarrow$
   $A \to \bullet\ z$
4. $A \to x\ y\ A\ \bullet$

State 3: Shifting the x will create a state just like state 1. So we recycle it. Note the “back arrow” in the state diagram.

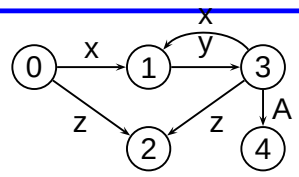
Consider the grammar $A \to x\ y\ A$, $A \to z$



0. $A \to \bullet\ x\ y\ A$
   $A \to \bullet z$
1. $A \to x\ \bullet\ y\ A$
2. $A \to z\ \bullet$
3. $A \to x\ y\ \bullet\ A$
   $A \to \bullet\ x\ y\ A$
   $A \to \bullet\ z \Leftarrow$
4. $A \to x\ y\ A\ \bullet$

State 3: Same situation with shifting z, only we recycle state 2.

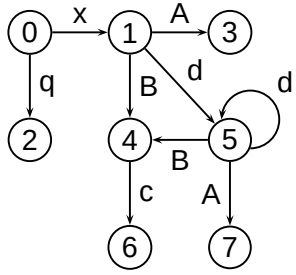
Consider the grammar $A \to x\ y\ A$, $A \to z$



0. $A \to \bullet\ x\ y\ A$
   $A \to \bullet z$
1. $A \to x\ \bullet\ y\ A$
2. $A \to z\ \bullet$
3. $A \to x\ y\ \bullet\ A$
   $A \to \bullet\ x\ y\ A$
   $A \to \bullet\ z$
4. $A \to x\ y\ A\ \bullet$ ⇐

State 4: No input expected. The automaton is complete.

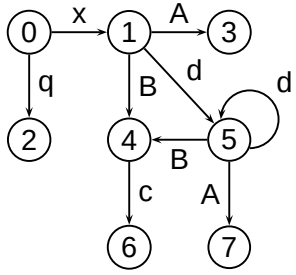
$S \to x\ A \mid q$
$A \to B\ c$
$B \to d\ A \mid d$



0. $S \to \bullet\ x\ A$
   $S \to \bullet\ q$

We have multiple productions this time. We only use the "start" symbol S.

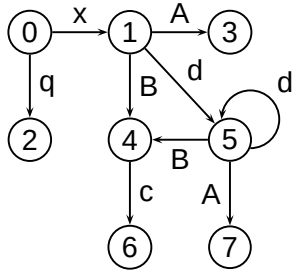
$S \to x\ A \mid q$
$A \to B\ c$
$B \to d\ A \mid d$



0. $S \to \bullet\ x\ A$ ⇐
   $S \to \bullet\ q$
1. $S \to x\ \bullet\ A$
   $A \to \bullet\ B\ c$
   $B \to \bullet\ d\ A$
   $B \to \bullet\ d$

State 0: shift x and take transitive closure to make state 1.

$S \to x\ A \mid q$
$A \to B\ c$
$B \to d\ A \mid d$



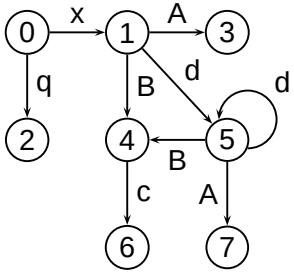
0. $S \to \bullet\ x\ A$
   $S \to \bullet\ q$ ⇐
1. $S \to x\ \bullet\ A$
   $A \to \bullet\ B\ c$
   $B \to \bullet\ d\ A$
   $B \to \bullet\ d$
2. $S \to q\ \bullet$

State 0: shift q to make state 2.

$S \rightarrow x\ A \mid q$

$A \rightarrow B\ c$

$B \rightarrow d\ A \mid d$



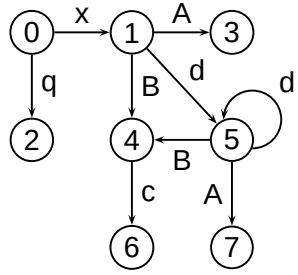
0. $S \rightarrow \bullet\ x\ A$
   $S \rightarrow \bullet\ q$
1. $S \rightarrow x\ \bullet\ A \Leftarrow$
   $A \rightarrow \bullet\ B\ c$
   $B \rightarrow \bullet\ d\ A$
   $B \rightarrow \bullet\ d$
2. $S \rightarrow q\ \bullet$
3. $S \rightarrow x\ A\ \bullet$

State 1: shift A to make state 3.

$S \rightarrow x\ A \mid q$

$A \rightarrow B\ c$

$B \rightarrow d\ A \mid d$



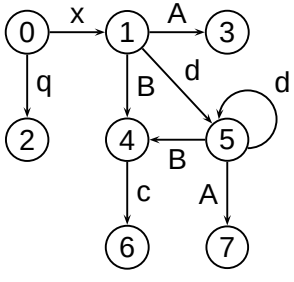
0. $S \rightarrow \bullet\ x\ A$
   $S \rightarrow \bullet\ q$
1. $S \rightarrow x\ \bullet\ A$
   $A \rightarrow \bullet\ B\ c \Leftarrow$
   $B \rightarrow \bullet\ d\ A$
   $B \rightarrow \bullet\ d$
2. $S \rightarrow q\ \bullet$
3. $S \rightarrow x\ A\ \bullet$
4. $A \rightarrow B\ \bullet\ c$

State 1: shift B to make state 4.

$S \rightarrow x\ A \mid q$

$A \rightarrow B\ c$

$B \rightarrow d\ A \mid d$



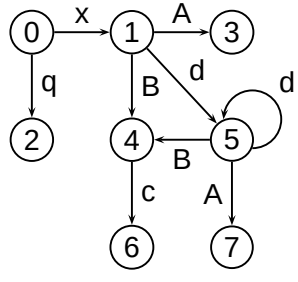
0. $S \rightarrow \bullet\ x\ A$
   $S \rightarrow \bullet\ q$
1. $S \rightarrow x\ \bullet\ A$
   $A \rightarrow \bullet\ B\ c$
   $B \rightarrow \bullet\ d\ A \Leftarrow$
   $B \rightarrow \bullet\ d \Leftarrow$
2. $S \rightarrow q\ \bullet$
3. $S \rightarrow x\ A\ \bullet$
4. $A \rightarrow B\ \bullet\ c$
5. $B \rightarrow d\ \bullet\ A$
   $B \rightarrow d\ \bullet$
   $A \rightarrow \bullet\ B\ c$
   $B \rightarrow \bullet\ d\ A$
   $B \rightarrow \bullet\ d$

State 1: shift d to make state 5. A lot happens here!

$S \rightarrow x\ A \mid q$

$A \rightarrow B\ c$

$B \rightarrow d\ A \mid d$



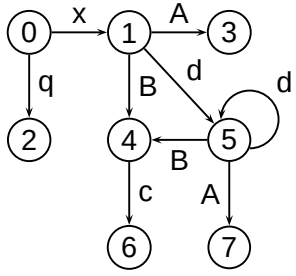
0. $S \rightarrow \bullet\ x\ A$
   $S \rightarrow \bullet\ q$
1. $S \rightarrow x\ \bullet\ A$
   $A \rightarrow \bullet\ B\ c$
   $B \rightarrow \bullet\ d\ A$
   $B \rightarrow \bullet\ d$
2. $S \rightarrow q\ \bullet$
3. $S \rightarrow x\ A\ \bullet$
4. $A \rightarrow B\ \bullet\ c \Leftarrow$
5. $B \rightarrow d\ \bullet\ A$
   $B \rightarrow d\ \bullet$
   $A \rightarrow \bullet\ B\ c$
   $B \rightarrow \bullet\ d\ A$
   $B \rightarrow \bullet\ d$
6. $A \rightarrow B\ c\ \bullet$

State 4: shift c to make state 6.

$S \rightarrow x\ A \mid q$

$A \rightarrow B\ c$

$B \rightarrow d\ A \mid d$



0. $S \rightarrow \bullet\ x\ A$
   $S \rightarrow \bullet\ q$
1. $S \rightarrow x\ \bullet\ A$
   $A \rightarrow \bullet\ B\ c$
   $B \rightarrow \bullet\ d\ A$
   $B \rightarrow \bullet\ d$
2. $S \rightarrow q\ \bullet$
3. $S \rightarrow x\ A\ \bullet$
4. $A \rightarrow B\ \bullet\ c$
5. $B \rightarrow d\ \bullet\ A \Leftarrow$
   $B \rightarrow d\ \bullet$
   $A \rightarrow \bullet\ B\ c$
   $B \rightarrow \bullet\ d\ A$
   $B \rightarrow \bullet\ d$
6. $A \rightarrow B\ c\ \bullet$
7. $B \rightarrow d\ A\ \bullet$

State 5: shift A to make state 7. The other shifts recycle. We are done.

Definition: "Create an initial item" for production $A \rightarrow \alpha$ results in an item $A \rightarrow \bullet\ \alpha$

1. Take all the productions for the initial state, create initial items for them, and put them into state 1.
2. For each advancable item i we've not yet considered:
   (a) Create a new state s. Suppose i's cursor will advance over x.
   (b) For each item j in the current state which advances over x, create new items j' by advancing the cursor, and add j' to s.
   (c) Take the transitive closure of s.
   (d) If s is already part of our set of states, discard it. Otherwise add it and continue with the next item.

Generate the Characteristic Finite State Machine for the following grammar.

$S \rightarrow b\ A\ e$

$A \rightarrow x\ E$

$A \rightarrow x\ I$

$E \rightarrow a\ N$

$I \rightarrow c\ N$

$N \rightarrow y$

Answer:

0. $S \rightarrow \bullet\ b\ A\ e$

Answer:

0. $S \rightarrow \bullet\ b\ A\ e \Leftarrow$
1. $S \rightarrow b\ \bullet\ A\ e$
   $A \rightarrow \bullet\ x\ E$
   $A \rightarrow \bullet\ x\ I$

LR Parsing
Answer
February 14, 2007
§4 **Activity**

Answer:

0. $S \rightarrow \bullet\ b\ A\ e$
1. $S \rightarrow b\ \bullet\ A\ e \Leftarrow$
   $A \rightarrow \bullet\ x\ E$
   $A \rightarrow \bullet\ x\ I$
2. $S \rightarrow b\ A\ \bullet\ e$

Illinois Institute of Technology Mattox Beckman
LR Parsing – p. 13

LR Parsing
Answer
February 14, 2007
§4 **Activity**

Answer:

0. $S \rightarrow \bullet\ b\ A\ e$
1. $S \rightarrow b\ \bullet\ A\ e$
   $A \rightarrow \bullet\ x\ E \Leftarrow$
   $A \rightarrow \bullet\ x\ I \Leftarrow$
2. $S \rightarrow b\ A\ \bullet\ e$
3. $A \rightarrow x\ \bullet\ E$
   $A \rightarrow x\ \bullet\ I$
   $E \rightarrow \bullet\ a\ N$
   $I \rightarrow \bullet\ c\ N$

Illinois Institute of Technology Mattox Beckman
LR Parsing – p. 13

LR Parsing
Answer
February 14, 2007
§4 **Activity**

Answer:

0. $S \rightarrow \bullet\ b\ A\ e$
1. $S \rightarrow b\ \bullet\ A\ e$
   $A \rightarrow \bullet\ x\ E$
   $A \rightarrow \bullet\ x\ I$
2. $S \rightarrow b\ A\ \bullet\ e \Leftarrow$
3. $A \rightarrow x\ \bullet\ E$
   $A \rightarrow x\ \bullet\ I$
   $E \rightarrow \bullet\ a\ N$
   $I \rightarrow \bullet\ c\ N$
4. $S \rightarrow b\ A\ e\ \bullet$

Illinois Institute of Technology Mattox Beckman
LR Parsing – p. 13

Answer:

0. $S \rightarrow \bullet\ b\ A\ e$
1. $S \rightarrow b\ \bullet\ A\ e$
   $A \rightarrow \bullet\ x\ E$
   $A \rightarrow \bullet\ x\ I$
2. $S \rightarrow b\ A\ \bullet\ e$
3. $A \rightarrow x\ \bullet\ E$ ⇐
   $A \rightarrow x\ \bullet\ I$
   $E \rightarrow \bullet\ a\ N$
   $I \rightarrow \bullet\ c\ N$
4. $S \rightarrow b\ A\ e\ \bullet$
5. $A \rightarrow x\ E\ \bullet$

LR Parsing
Answer
February 14, 2007
§4 **Activity**

Answer:

0. $S \rightarrow \bullet\ b\ A\ e$
1. $S \rightarrow b\ \bullet\ A\ e$
   $A \rightarrow \bullet\ x\ E$
   $A \rightarrow \bullet\ x\ I$
2. $S \rightarrow b\ A\ \bullet\ e$
3. $A \rightarrow x\ \bullet\ E$
   $A \rightarrow x\ \bullet\ I$ ⇐
   $E \rightarrow \bullet\ a\ N$
   $I \rightarrow \bullet\ c\ N$
4. $S \rightarrow b\ A\ e\ \bullet$
5. $A \rightarrow x\ E\ \bullet$
6. $A \rightarrow x\ I\ \bullet$

Illinois Institute of Technology Mattox Beckman
LR Parsing – p. 13

LR Parsing
Answer
February 14, 2007
§4 **Activity**

Answer:

0. $S \rightarrow \bullet\ b\ A\ e$
1. $S \rightarrow b\ \bullet\ A\ e$
   $A \rightarrow \bullet\ x\ E$
   $A \rightarrow \bullet\ x\ I$
2. $S \rightarrow b\ A\ \bullet\ e$
3. $A \rightarrow x\ \bullet\ E$
   $A \rightarrow x\ \bullet\ I$
   $E \rightarrow \bullet\ a\ N$ ⇐
   $I \rightarrow \bullet\ c\ N$
4. $S \rightarrow b\ A\ e\ \bullet$
5. $A \rightarrow x\ E\ \bullet$
6. $A \rightarrow x\ I\ \bullet$
7. $E \rightarrow a\ \bullet\ N$
   $N \rightarrow \bullet\ y$

Illinois Institute of Technology Mattox Beckman
LR Parsing – p. 13

LR Parsing
Answer
February 14, 2007
§4 **Activity**

Answer:

0. $S \rightarrow \bullet\ b\ A\ e$
1. $S \rightarrow b\ \bullet\ A\ e$
   $A \rightarrow \bullet\ x\ E$
   $A \rightarrow \bullet\ x\ I$
2. $S \rightarrow b\ A\ \bullet\ e$
3. $A \rightarrow x\ \bullet\ E$
   $A \rightarrow x\ \bullet\ I$
   $E \rightarrow \bullet\ a\ N$
   $I \rightarrow \bullet\ c\ N$ ⇐
4. $S \rightarrow b\ A\ e\ \bullet$
5. $A \rightarrow x\ E\ \bullet$
6. $A \rightarrow x\ I\ \bullet$
7. $E \rightarrow a\ \bullet\ N$
   $N \rightarrow \bullet\ y$
8. $I \rightarrow c\ \bullet\ N$
   $N \rightarrow \bullet\ y$

Illinois Institute of Technology Mattox Beckman
LR Parsing – p. 13

Answer:

0. $S \rightarrow \bullet\ b\ A\ e$
1. $S \rightarrow b\ \bullet\ A\ e$
   $A \rightarrow \bullet\ x\ E$
   $A \rightarrow \bullet\ x\ I$
2. $S \rightarrow b\ A\ \bullet\ e$
3. $A \rightarrow x\ \bullet\ E$
   $A \rightarrow x\ \bullet\ I$
   $E \rightarrow \bullet\ a\ N$
   $I \rightarrow \bullet\ c\ N$
4. $S \rightarrow b\ A\ e\ \bullet$
5. $A \rightarrow x\ E\ \bullet$
6. $A \rightarrow x\ I\ \bullet$
7. $E \rightarrow a\ \bullet\ N$ ⇐
   $N \rightarrow \bullet\ y$
8. $I \rightarrow c\ \bullet\ N$
   $N \rightarrow \bullet\ y$
9. $E \rightarrow a\ N\ \bullet$

LR Parsing
Answer
February 14, 2007
§4 **Activity**

Answer:

0. $S \rightarrow \bullet\ b\ A\ e$
1. $S \rightarrow b\ \bullet\ A\ e$
   $A \rightarrow \bullet\ x\ E$
   $A \rightarrow \bullet\ x\ I$
2. $S \rightarrow b\ A\ \bullet\ e$
3. $A \rightarrow x\ \bullet\ E$
   $A \rightarrow x\ \bullet\ I$
   $E \rightarrow \bullet\ a\ N$
   $I \rightarrow \bullet\ c\ N$
4. $S \rightarrow b\ A\ e\ \bullet$
5. $A \rightarrow x\ E\ \bullet$
6. $A \rightarrow x\ I\ \bullet$
7. $E \rightarrow a\ \bullet\ N$
   $N \rightarrow \bullet\ y$ ⇐
8. $I \rightarrow c\ \bullet\ N$
   $N \rightarrow \bullet\ y$
9. $E \rightarrow a\ N\ \bullet$
10. $N \rightarrow y\ \bullet$

Illinois Institute of Technology Mattox Beckman
LR Parsing – p. 13

LR Parsing
Answer
February 14, 2007
§4 **Activity**

Answer:

0. $S \rightarrow \bullet\ b\ A\ e$
1. $S \rightarrow b\ \bullet\ A\ e$
   $A \rightarrow \bullet\ x\ E$
   $A \rightarrow \bullet\ x\ I$
2. $S \rightarrow b\ A\ \bullet\ e$
3. $A \rightarrow x\ \bullet\ E$
   $A \rightarrow x\ \bullet\ I$
   $E \rightarrow \bullet\ a\ N$
   $I \rightarrow \bullet\ c\ N$
4. $S \rightarrow b\ A\ e\ \bullet$
5. $A \rightarrow x\ E\ \bullet$
6. $A \rightarrow x\ I\ \bullet$
7. $E \rightarrow a\ \bullet\ N$
   $N \rightarrow \bullet\ y$
8. $I \rightarrow c\ \bullet\ N$ ⇐
   $N \rightarrow \bullet\ y$
9. $E \rightarrow a\ N\ \bullet$
10. $N \rightarrow y\ \bullet$
11. $I \rightarrow c\ N\ \bullet$

Illinois Institute of Technology Mattox Beckman
LR Parsing – p. 13

Example Grammar: $E \rightarrow E\ +\ T \mid T$
$T \rightarrow T\ * \mid \texttt{int}$

Here is the CFSM:

I_0 $E \rightarrow \bullet\ E + T$
$E \rightarrow \bullet\ T$
$T \rightarrow \bullet\ T\ *$
$T \rightarrow \bullet\ \texttt{int}$

I_1 $E \rightarrow E\ \bullet\ + T$

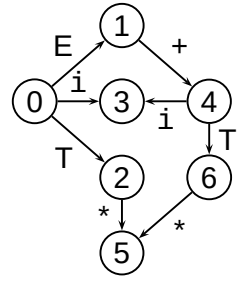
I_2 $E \rightarrow T\ \bullet$
$T \rightarrow T\ \bullet\ *$

I_3 $T \rightarrow \texttt{int}\ \bullet$

I_4 $E \rightarrow E + \bullet\ T$
$T \rightarrow \bullet\ T\ *$
$T \rightarrow \bullet\ \texttt{int}$

I_5 $T \rightarrow T\ *\ \bullet$

I_6 $E \rightarrow E + T\ \bullet$
$T \rightarrow T\ \bullet\ *$



The LR Tables

- In order to use this in a real parse, you need an extra piece of information: the LR tables.

1. $E \rightarrow E + T$
2. $E \rightarrow T$
3. $T \rightarrow T *$
4. $T \rightarrow$ `int`

	Action				Go To	
State	+	*	int	$	E	T
0			S 3		1	2
1	S 4					
2	R 2	S 5		R 2		
3	R 4	R 4		R 4		
4			S 3			6
5	R 3	R 3		R 3		
6	R 1	S 5		R 1		

Decoding the Table

- "S n" means to shift the token and go to state n.
- "R n" means to reduce by rule n.
- "n" by itself means to go to state n.
- A blank entry indicates an error.
- Given the next input, check the table to see what action you need to take.
- When shifting, push the current state and the new token onto the stack.
- When reducing, pop the appropriate number of elements and states off the stack and construct a new element via the rule. Put the new element back on the stack.

Example

Action: 0 — Shift 3
State Stack: 0
Symbol Stack:

Input: `int * + int + int *`

	Action				Go To	
State	+	*	int	$	E	T
0			S 3		1	2
1	S 4					
2	R 2	S 5		R 2		
3	R 4	R 4		R 4		
4			S 3			6
5	R 3	R 3		R 3		
6	R 1	S 5		R 1		

Example

Action: 3 — Reduce 4
State Stack: 3 0
Symbol Stack:
`int`

Input: `* + int + int *`

	Action				Go To	
State	+	*	int	$	E	T
0			S 3		1	2
1	S 4					
2	R 2	S 5		R 2		
3	R 4	R 4		R 4		
4			S 3			6
5	R 3	R 3		R 3		
6	R 1	S 5		R 1		

Action: 0 — Goto 2
State Stack: 0
Symbol Stack:

Input: `T * + int + int *`

	Action				Go To	
State	+	*	int	$	E	T
0			S 3		1	2
1	S 4					
2	R 2	S 5		R 2		
3	R 4	R 4		R 4		
4			S 3			6
5	R 3	R 3		R 3		
6	R 1	S 5		R 1		

Action: 2 — Shift 5
State Stack: 2 0
Symbol Stack:

Input: `* + int + int *`

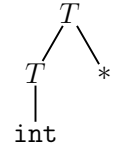
	Action				Go To	
State	+	*	int	$	E	T
0			S 3		1	2
1	S 4					
2	R 2	S 5		R 2		
3	R 4	R 4		R 4		
4			S 3			6
5	R 3	R 3		R 3		
6	R 1	S 5		R 1		

Action: 5 — Reduce 3
State Stack: 5 2 0
Symbol Stack:

Input: `+ int + int *`

	Action				Go To	
State	+	*	int	$	E	T
0			S 3		1	2
1	S 4					
2	R 2	S 5		R 2		
3	R 4	R 4		R 4		
4			S 3			6
5	R 3	R 3		R 3		
6	R 1	S 5		R 1		

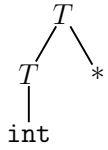
Action: 0 — Goto 2
State Stack: 0
Symbol Stack:



Input: `T + int + int *`

	Action				Go To	
State	+	*	int	$	E	T
0			S 3		1	2
1	S 4					
2	R 2	S 5		R 2		
3	R 4	R 4		R 4		
4			S 3			6
5	R 3	R 3		R 3		
6	R 1	S 5		R 1		

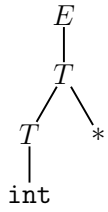
Action: 2 — Reduce 2
State Stack: 2 0
Symbol Stack:



Input: `+ int + int *`

	Action				Go To	
State	+	*	int	$	E	T
0			S 3		1	2
1	S 4					
2	R 2	S 5		R 2		
3	R 4	R 4		R 4		
4			S 3			6
5	R 3	R 3		R 3		
6	R 1	S 5		R 1		

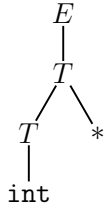
Action: 0 — Goto 1
State Stack: 0
Symbol Stack:



Input: `E + int + int *`

	Action				Go To	
State	+	*	int	$	E	T
0			S 3		1	2
1	S 4					
2	R 2	S 5		R 2		
3	R 4	R 4		R 4		
4			S 3			6
5	R 3	R 3		R 3		
6	R 1	S 5		R 1		

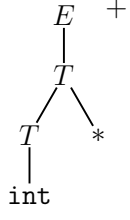
Action: 1 — Shift 4
State Stack: 1 0
Symbol Stack:



Input: `+ int + int *`

	Action				Go To	
State	+	*	int	$	E	T
0			S 3		1	2
1	S 4					
2	R 2	S 5		R 2		
3	R 4	R 4		R 4		
4			S 3			6
5	R 3	R 3		R 3		
6	R 1	S 5		R 1		

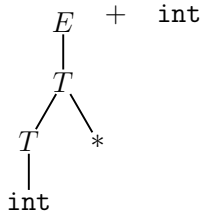
Action: 4 — Shift 3
State Stack: 4 1 0
Symbol Stack:



Input: `int + int *`

	Action				Go To	
State	+	*	int	$	E	T
0			S 3		1	2
1	S 4					
2	R 2	S 5		R 2		
3	R 4	R 4		R 4		
4			S 3			6
5	R 3	R 3		R 3		
6	R 1	S 5		R 1		

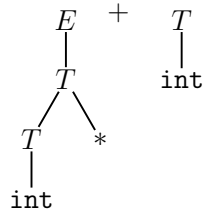
Action: 3 — Reduce 4
State Stack: 3 4 1 0
Symbol Stack:



Input: `+ int *`

	Action				Go To	
State	+	*	int	$	E	T
0			S 3		1	2
1	S 4					
2	R 2	S 5		R 2		
3	R 4	R 4		R 4		
4			S 3			6
5	R 3	R 3		R 3		
6	R 1	S 5		R 1		

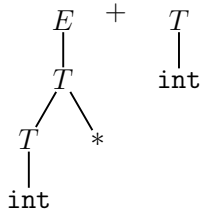
Action: 4 — Goto 6
State Stack: 4 1 0
Symbol Stack:



Input: `T + int *`

	Action				Go To	
State	+	*	int	$	E	T
0			S 3		1	2
1	S 4					
2	R 2	S 5		R 2		
3	R 4	R 4		R 4		
4			S 3			6
5	R 3	R 3		R 3		
6	R 1	S 5		R 1		

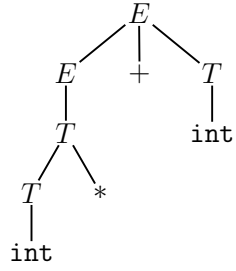
Action: 6 — Reduce 1
State Stack: 6 4 1 0
Symbol Stack:



Input: `+ int *`

	Action				Go To	
State	+	*	int	$	E	T
0			S 3		1	2
1	S 4					
2	R 2	S 5		R 2		
3	R 4	R 4		R 4		
4			S 3			6
5	R 3	R 3		R 3		
6	R 1	S 5		R 1		

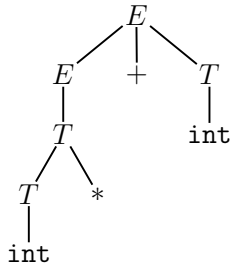
Action: 0 — Goto 1
State Stack: 0
Symbol Stack:



Input: `E + int *`

	Action				Go To	
State	+	*	int	$	E	T
0			S 3		1	2
1	S 4					
2	R 2	S 5		R 2		
3	R 4	R 4		R 4		
4			S 3			6
5	R 3	R 3		R 3		
6	R 1	S 5		R 1		

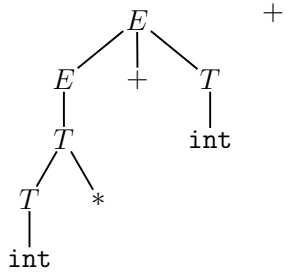
Action: 1 — Shift 4
State Stack: 1 0
Symbol Stack:



Input: `+ int *`

	Action				Go To	
State	+	*	int	$	E	T
0			S 3		1	2
1	S 4					
2	R 2	S 5		R 2		
3	R 4	R 4		R 4		
4			S 3			6
5	R 3	R 3		R 3		
6	R 1	S 5		R 1		

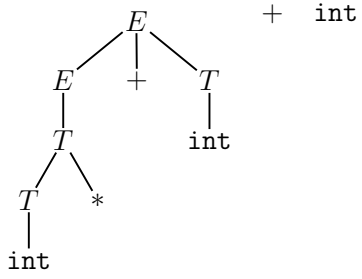
Action: 4 — Shift 3
State Stack: 4 1 0
Symbol Stack:



Input: `int *`

	Action				Go To	
State	+	*	int	$	E	T
0			S 3		1	2
1	S 4					
2	R 2	S 5		R 2		
3	R 4	R 4		R 4		
4			S 3			6
5	R 3	R 3		R 3		
6	R 1	S 5		R 1		

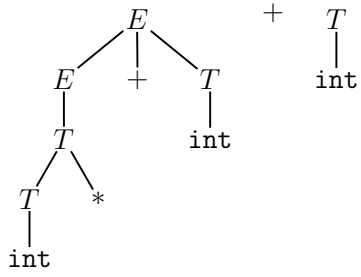
Action: 3 — Reduce 4
State Stack: 3 4 1 0
Symbol Stack:



Input: `*`

	Action				Go To	
State	+	*	int	$	E	T
0			S 3		1	2
1	S 4					
2	R 2	S 5		R 2		
3	R 4	R 4		R 4		
4			S 3			6
5	R 3	R 3		R 3		
6	R 1	S 5		R 1		

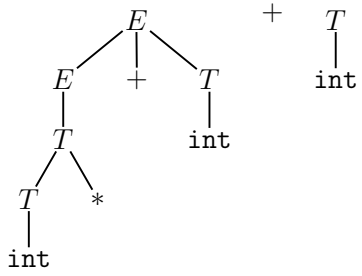
Action: 4 — Goto 6
State Stack: 4 1 0
Symbol Stack:



Input: `T *`

	Action				Go To	
State	+	*	int	$	E	T
0			S 3		1	2
1	S 4					
2	R 2	S 5		R 2		
3	R 4	R 4		R 4		
4			S 3			6
5	R 3	R 3		R 3		
6	R 1	S 5		R 1		

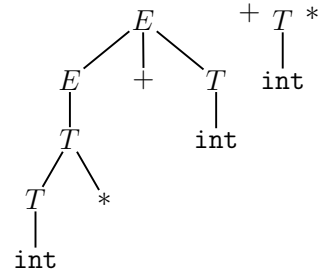
Action: 6 — Shift 5
State Stack: 6 4 1 0
Symbol Stack:



Input: *

	Action				Go To	
State	+	*	int	$	E	T
0			S 3		1	2
1	S 4					
2	R 2	S 5		R 2		
3	R 4	R 4		R 4		
4			S 3			6
5	R 3	R 3		R 3		
6	R 1	S 5		R 1		

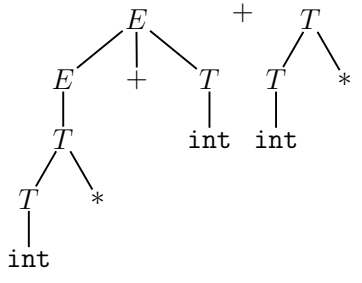
Action: 5 — Reduce 3
State Stack: 5 6 4 1 0
Symbol Stack:



Input:

	Action				Go To	
State	+	*	int	$	E	T
0			S 3		1	2
1	S 4					
2	R 2	S 5		R 2		
3	R 4	R 4		R 4		
4			S 3			6
5	R 3	R 3		R 3		
6	R 1	S 5		R 1		

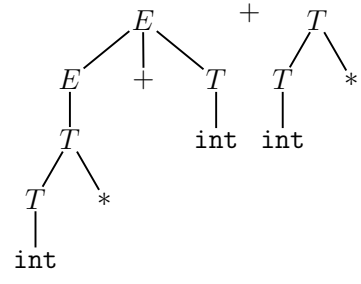
Action: 4 — Goto 6
State Stack: 4 1 0
Symbol Stack:



Input: T

	Action				Go To	
State	+	*	int	$	E	T
0			S 3		1	2
1	S 4					
2	R 2	S 5		R 2		
3	R 4	R 4		R 4		
4			S 3			6
5	R 3	R 3		R 3		
6	R 1	S 5		R 1		

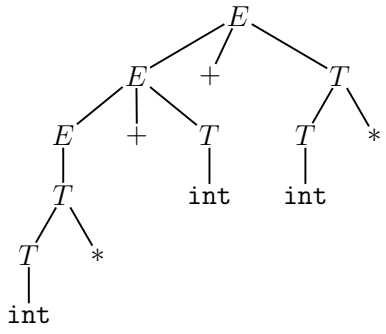
Action: 6 — Reduce 1
State Stack: 6 4 1 0
Symbol Stack:



Input:

	Action				Go To	
State	+	*	int	$	E	T
0			S 3		1	2
1	S 4					
2	R 2	S 5		R 2		
3	R 4	R 4		R 4		
4			S 3			6
5	R 3	R 3		R 3		
6	R 1	S 5		R 1		

Action: 0 — Accept
State Stack: 0
Symbol Stack:



Input:

	Action				Go To	
State	+	*	int	$	E	T
0			S 3		1	2
1	S 4					
2	R 2	S 5		R 2		
3	R 4	R 4		R 4		
4			S 3			6
5	R 3	R 3		R 3		
6	R 1	S 5		R 1		

- First, generate the CFSM.
- Add a shift or goto operation for each transition in the machine.
- When the • is at the end of a production, you will add a reduce rule... but where?

	Action				Go To	
State	+	*	int	$	E	T
0			S 3		1	2
1	S 4					
2	R 2	S 5		R 2		
3	R 4	R 4		R 4		

...

I_2 $E \rightarrow T \bullet$
$T \rightarrow T \bullet *$

In other words, how does state 2 of this machine know what to do?

It's easiest to make the tables while you are making the CFSM.

1. Make the First and Follow sets for the nonterminals.
2. As you make the CFSM:
   (a) Looking at a terminal? Add a *shift* operation for that terminal in the Action table. Put the destination state in the Goto table.
   (b) At the end of a rule $X \rightarrow Y$? Add a *reduce* operation for each terminal in the Follow set of X.
   (c) Looking at a non-terminal? Add the destination state to the Goto table.

1. $E \rightarrow E + T$
2. $E \rightarrow T$
3. $T \rightarrow T *$
4. $T \rightarrow$ `int`

State	+	*	int	$
E				X
T				

Make the table, and add $ to Follow(E).

Example — Make Follow

1. $E \rightarrow E + T$
2. $E \rightarrow T$
3. $T \rightarrow T *$
4. $T \rightarrow$ `int`

State	+	*	int	$
E	X			X
T		X		

Check productions: add + to Follow(E) and * to Follow(T).

Example — Make Follow

1. $E \rightarrow E + T$
2. $E \rightarrow T$
3. $T \rightarrow T *$
4. $T \rightarrow$ `int`

State	+	*	int	$
E	X			X
T	X	X		X

Check endings: E ends with T, so add Follow(E) to Follow(T).

Example — Make LR Table

I_0 $E \rightarrow \bullet\, E + T \mid \bullet\, T$
$T \rightarrow \bullet\, T * \mid \bullet$ `int`
I_1 $E \rightarrow E \bullet + T$
I_2 $E \rightarrow T \bullet$
$T \rightarrow T \bullet *$
I_3 $T \rightarrow$ `int` $\bullet$
I_4 $E \rightarrow E + \bullet\, T$
$T \rightarrow \bullet\, T *$
$T \rightarrow \bullet$ `int`
I_5 $T \rightarrow T * \bullet$
I_6 $E \rightarrow E + T \bullet$
$T \rightarrow T \bullet *$

	Action				Go To	
State	+	*	int	$	E	T
0			S 3		1	2
1	S 4					
2	R 2	S 5		R 2		
3	R 4	R 4		R 4		
4			S 3			6
5	R 3	R 3		R 3		
6	R 1	S 5		R 1		

Consider this grammar:

$E \rightarrow$ `if` E `then` $E \mid$ `if` E `then` E `else` E

First(E) = {`if`} Follow(E) = {`then`, `else`, $}

1. $E \rightarrow \bullet$ `if` E `then` $E \mid \bullet$ `if` E `then` E `else` E
2. $E \rightarrow$ `if` $\bullet\, E$ `then` $E \mid$ `if` $\bullet\, E$ `then` E `else` E
   $E \rightarrow \bullet$ `if` E `then` $E \mid \bullet$ `if` E `then` E `else` E
3. $E \rightarrow$ `if` $E \bullet$ `then` $E \mid$ `if` $E \bullet$ `then` E `else` E
4. $E \rightarrow$ `if` E `then` $\bullet\, E \mid$ `if` E `then` $\bullet\, E$ `else` E
   $E \rightarrow \bullet$ `if` E `then` $E \mid \bullet$ `if` E `then` E `else` E
5. $E \rightarrow$ `if` E `then` $E \bullet \mid$ `if` E `then` $E \bullet$ `else` E

Activity! Make the LR Table, and think about what should happen at state 5.

Here is the CFSM:

1. $E \rightarrow \bullet$ `if` E `then` $E \mid \bullet$ `if` E `then` E `else` E
2. $E \rightarrow$ `if` $\bullet$ E `then` $E \mid$ `if` $\bullet$ E `then` E `else` E
   $E \rightarrow \bullet$ `if` E `then` $E \mid \bullet$ `if` E `then` E `else` E
3. $E \rightarrow$ `if` $E \bullet$ `then` $E \mid$ `if` $E \bullet$ `then` E `else` E
4. $E \rightarrow$ `if` E `then` $\bullet$ $E \mid$ `if` E `then` $\bullet$ E `else` E
   $E \rightarrow \bullet$ `if` E `then` $E \mid \bullet$ `if` E `then` E `else` E
5. $E \rightarrow$ `if` E `then` $E \bullet \mid$ `if` E `then` $E \bullet$ `else` E
6. $E \rightarrow$ `if` E `then` E `else` $\bullet$ E
   $E \rightarrow \bullet$ `if` E `then` $E \mid \bullet$ `if` E `then` E `else` E
7. $E \rightarrow$ `if` E `then` E `else` $E \bullet$

Here is the LR table:

Action Table

State	`if`	`then`	`else`	$	E
1	S 2				
2	S 2				3
3		S 4			
4	S 2				5
5		R 1	S 6 / R 1	R 1	
6	S 2				7
7		R 2	R 2	R 2	

There are two ways that shift-reduce conflicts are usually handled.

1. Make a *precedence* rule. E.g.

   $$E \rightarrow E \bullet + E$$
   $$E \rightarrow E * E \bullet$$

   In this example, since * has higher precedence, the reducing rule will be used, and the shifting rule ignored.
2. When in doubt, shift. E.g.

   $$E \rightarrow \texttt{if}\ E\ \texttt{then}\ E \bullet$$
   $$E \rightarrow \texttt{if}\ E\ \texttt{then}\ E \bullet\ \texttt{else}\ E$$

$S \rightarrow A \mid B$
$A \rightarrow x\ y$
$B \rightarrow x\ y$

What will happen in the CFSM?

I_0: $S \rightarrow \bullet A$, $S \rightarrow \bullet B$, $A \rightarrow \bullet x\ y$, $B \rightarrow \bullet x\ y$

I_1: $S \rightarrow A \bullet$

I_2: $S \rightarrow B \bullet$

I_3: $A \rightarrow x \bullet y$, $B \rightarrow x \bullet y$

I_4: $A \rightarrow x\ y \bullet$, $B \rightarrow x\ y \bullet$

State 4 has a reduce-reduce conflict. This grammar has serious problems.